Remote Monitoring System

CPEG-597-Masters Project

Submitted to: ***Prof. Gohnsin Liu***

Submitted by : ***Sreekanth Lanke****: ID-* ***641542***
Date of submission: December ,14, 2005.
Department Of Computer Engineering.

Acknowledgments

It is my privilege to offer a deep sense of gratitude and thanks to **Prof. Gohnsin Liu** my project guide , for his valuable guidance , advice and constant encouragement and supervision throughout the course of my project work and who has also been a source inspiration and motivation thru all stages of this project .

I would also like to express my sincere thanks to **Prof. Stephen Grodzinsky**, Chairman of the Dept. Computer Engineering for encouraging me to take guidance from Prof. Gohnsin Liu.

I would like to thank my firm for providing all infrastructure facilities required to test my project . Hence this acknowledgement is a humble attempt to earnestly thank those who were directly or indirectly involved in the project and were of immense help to us.

Index

6. SOFTWARE DESIGN DOCUMENT

7. PSEUDO CODE

ABSTRACT

The need for a comprehensive Remote Monitoring System in the Educational and Office Environments has been the source of inspiration for this project. The Project has been conceived and developed to work effectively in the above mentioned environments. The main aim was to reduce the burden on the System/Network administrators from responsibility of monitoring the Clients connected to a Server in the typical Ethernet based Client – Sever network architecture.

The Remote Monitoring System (RMS) monitors the Clients connected to the Server by providing information such as Client's Name, IP address, User currently logged on and the Desktop Image of the Client. RMS also provides the functionalities such as Messaging and Control operations that can be performed on the Clients. The Supervisor can use these features to effectively monitor the Clients.

The Architecture of Remote Monitoring System has two components:

- *An Application which runs on the Server machine.*
- *A DLL which resides on the Client Machines.*

The Graphical User Interface based Application acts as the monitoring tool and provides the various functionalities to monitor the Clients.

The DLL which resides on the Client machines sends required information as per the need to the Application at the server.

Rapid Application Development paradigm of Software Engineering has been used to develop the system. The system's development has been done based on the Application Programming Interface (API) of the Windows Environment, so that the system complies with all existing flavors of the Microsoft's Windows Operating Systems.

Rational Rose tool is used for analysis and design using Unified Modeling Language and the platform of implementation being Visual Studio 6.0.

1. INTRODUCTION

1.1 Introduction:

This Section describes the background of the system developed i.e. the need for such a system, its overview and scope. It also discusses few popular Remote Tools and their comparison to the developed Remote Monitoring System (RMS).

1.2 Need for the Remote Monitoring System:

The advent of Computer Network in the Industrial and Research, Educational fields has increased the scope for exploiting common pool of resources in a distributed fashion. This introduction of Computer Networks has the issues of Administration & Monitoring attached with it particularly in the Educational Institutions, Office Environments.

RMS has been conceived to relieve the burden of System/Network Administrator from monitoring the clients connected to a Network/LAN especially in Educational and Office Environments.

1.3 Over View:

The Remote Monitoring System (RMS) has been developed to monitor the Clients in Network Environment.

The Remote Monitoring System provides the supervisor with a Graphical User Interface based Front End with various features to utilize the package effectively.

The features of RMS are:

- Display of Name and IP address of a Client.
- Selecting a Client for monitoring the display/GUI, Transaction Time to Time.
- After the selection of the terminal, control functions like Shutdown, Reset can be executed from Remote.
- Further on the events of exigency if the supervisor wants to bring to the notice of Client about the severity of the problem he/she can prompt a message on the Client's display along with an audio beep This feature helps system level supervisor in the Process Industry, Power Plants, Cyber surveillance and in many networking environments.

- In the Educational Institutes and Campus Environments it enables a better control on the student PC to avoid him/Warning on the event of entering in to forbidden sites.

1.4 Scope of RMS:

RMS is a tool specifically developed to monitor the Clients connected to the Server remotely. It does not provide remote control functionality such as access to the File System or any other feature which gives control over the remote system.

RMS achieves its goal of *monitoring* by getting some system related information of the Clients and also the Desktop Image of the Client Systems.

It does not in any way disturb the Users working on the Remote Clients. Thus it works only as a Remote Monitoring Tool but not as a Remote Control Tool.

1.5 Existing systems:

In this Section we will present three of the most popular Remote Access Tools.

- Microsoft's Remote Desktop.
- Microsoft's Remote Assistance.
- NetSupport Manager V9.00.

1.5.1 Microsoft's Remote Desktop:

Remote Desktop which was previously known as Microsoft's Terminal Services client, provides a connection from your home computer to access a Remote computer.

With Remote Desktop, one can have access to a Windows session that is running on home computer when he/she is at remote computer. This means, for example, that you can connect to your work computer from home and have access to all of your applications, files, and network resources as though you were in front of your computer at work. You can leave programs running at work and when you get home you can see your desktop at work displayed on your home computer, with the same programs running.

1.5.2 Microsoft's Remote Assistance:

Microsoft's Remote Assistance is used to allow some remote user to connect to your home computer over the Internet, chat with you, and observe your computer screen as you work. With your permission, he or she can use your keyboard and mouse as you work together to solve your problem. This tool is specifically used to get assistance from Technical Support Staff to assist the home computer user remotely.

1.5.3 Net Support Manager:

NetSupport Manager combines PC remote control with desktop management functionality leading to one of the fastest levels of ROI available on the market today. This tool is used to control terminals connected in a work group or in a Domain by providing various features such as Monitoring i.e. viewing the Client desktop Images, Remote Control and Sharing of peripherals of the Clients.

1.6 Existing systems Vs Remote Monitoring System (RMS):

As mentioned earlier the RMS is only a Monitoring Tool and does not attempt to gain control over the Client systems.

The major comparisons between all the above mentioned tools are:

	RMS	Remote Desktop	Remote Assistance	NetSupport Manager
Objective	Remote Monitoring	Remote working Environment	Remote Technical Support	Remote Control
Platform	Works on all Windows Operating Systems.	Requires Windows XP professional or higher	Requires Windows XP professional or higher	Cross Platform.
Communication	LAN	Internet	Internet	LAN
Cost	Low Cost	O.S provided Utility	O.S provided Utility	High Cost. As it is a third party Tool.

Thus it can be inferred from the above table that:

- RMS is custom built as a monitoring tool,
- RMS works on all windows Operating Systems,
- RMS is a *low cost* system providing efficient *Monitoring* of the Clients.

BASICS OF COMMUNICATION

2. BASICS OF NETWORK COMMUNICATIONS

2.1 Introduction:

This Section is intended to discuss the basic concepts of Computer Networks and Network Communications.

2.2 Open Systems Interconnect:

Open Systems Interconnection (OSI) Reference model explains the basic layers of communication.

OSI model was developed based on the proposal of ISO.

The OSI model consists of 7 layers:

1. Physical Layer
2. Data Link Layer
3. Network Layer
4. Transport Layer
5. Session Layer
6. Presentation Layer
7. Application Layer.

Physical Layer:

Physical Layer deals with transportation of *raw bits* over communication channel.

Data Link Layer:

The basic of Data Link Layer is to transforms a raw transmission facility to a line that appears free of undetected transmission errors to the network *layer*.

Network Layer:

The Network Layer controls the operation of the subnet; a key design issue is determining how packets are routed from source to destination.

Transport Layer:

The basic function of the transport layer is to access data from above split-it up into smaller units if need be, pass these to the *network layer* and ensure that the pieces all arrive correctly the other end.

Session Layer:

The Session Layer allows users on different machine to establish Sessions between them.

Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

Fig: OSI Reference Model

Presentation Layer:

The Presentation Layer is connected with the syntax and semantics of the information transmitted.

Application Layer:

The Application Layer contains a variety of protocols that are commonly needed by users. (HTTP, FTP).

2.3 TCP/IP Model:

The TCP/IP Model is the most popular reference model which consists 4 layers.

1. Link Layer
2. Network Layer
3. Transport Layer
4. Application Layer

Link Layer (or) Network Reference Layer:

The Link Layer normally includes the device drivers in the operating system and the corresponding network interface card in the computer, together they handle all the hardware details of physically interfacing with the cable.

Network Layer:

The Network Layer handles the movement of packets around the network i.e. Routing of packets takes place here.

Transport Layer:

The Transport Layer provides a flow of data between two hosts for the application layer above it. *It provides two protocols*.

1 Transmission control protocol (TCP), which is reliable connection, oriented.

2 User Data gram Protocol (UDP) that is unreliable, connection less

Application Layer:

The Application Layer handles the details of the particular application (Telnet, FTP, and SMTP) etc...

2.4 Header:

In a transmission packet, the header contains controls and other information that precedes the data in a packet. A header field includes source and destination addresses, packet type information various types of identifier information and so on.

In addition to header and data portions, a packet may also have a trailer section after the data. The trailer generally includes error detection fields such as CRC.

In an E-mail message, the header is the information that provides the actual message. The message header includes information such as sender address, message subject, date and time.

2.5 Packet:

A common term for a standard unit of data sends across a network.

A packet is a well-defined block of bytes, which consists of header data and trailer. In a layered network architecture packets created at one level may be inserted into another header/trailer envelope at a lower level. Packets can be transmitted across a network or over telephone lines.

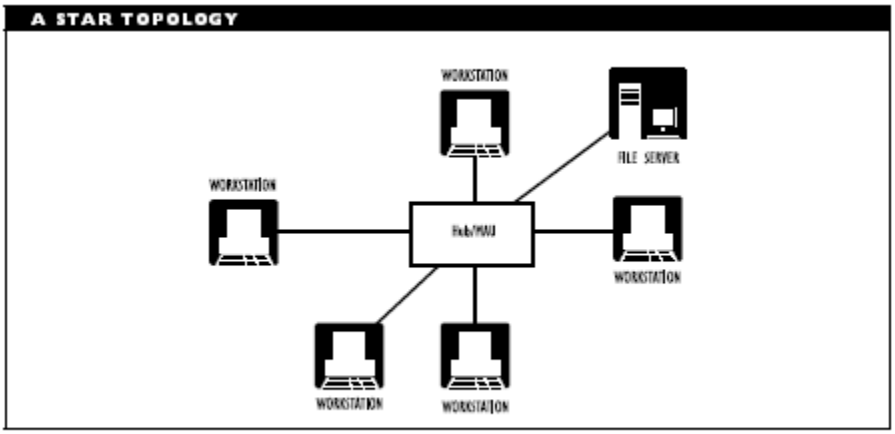
In fact in network protocols and serial communications protocols use packet switching to establish a connection and route information.

The format of packet depends on the protocol that creates the packet. Various communications standards and protocols use special purpose or specially defined packet to control or monitor a communication session.

Packets are sometimes also known as frames although that term originally referred specifically to a packet at the data link layer in the OSI reference model.

2.6 Star topology:

A star topology is a physical topology in which multiple nodes are connected to a central component generally known as a hub.



Despite appearance, such a winding scheme actually implements a logical bus topology.

A hub of a star generally is just a wiring center that is a, termination point for there nodes a single connection containing from the hub.

In rare cases the hub may actually be a file server, with all it's directly attached to the server.

All signals instructions and data going to and from each node must pass through the hub to which the node is connected.

Example the telephone company, wiring system with lines to individual subscriber coming from a central location.

Advantage of star topology is the trouble shooting and fault isolation is easy. Disadvantage is that if the hub fails, the entire network fails.

2.7 Cluster**:**

In a network particularly in a mainframe based network, a group of I/O devices such as terminals, computers or printers that share, common communication path to a host machine.

Communication between the device in a cluster, and host are generally managed by a cluster such as IBM 3274 controller.

2.8 File Transfer Protocol **(**FTP**)**:

File Transfer Protocol (FTP) is a method of transforming files from one computer to another. The computer that is requesting for a file is the FTP client, the Computer which services the requests is the FTP server and both of them must follow the FTP protocol.

There are millions of useful and important files stored on Internet like software manuals, shareware programs, most of which are kept at FTP sites. Many of FTP sites have a public Area, which can be accessed by any body with the Internet connection but the other directories and the files stored in them are only for authorized members only.The service offered by such sites that allow anybody to access them is called anonymous (FTP).

Advantages:

FTP was and still is in most cases the fastest and easiest way to transfer a file from one machine to another over the Network.

2.9 Hyper Text Transfer Protocol (HTTP):

The Hyper Text Transfer protocol used throughout the World Wide Web (WWW) is HTTP. It specifies what messages clients may send to servers and what responses they get back in return.

Each interaction consists of one ASCII request, followed by one RFC 822MIME like response.

All clients and all servers must obey this protocol. It is desired in RFC 2616.So HTTP is mainly used for sending and Reading web pages.

2.10 Hyper Text Markup Language (HTML):

The Hyper Text Markup Language is the one where the web pages are currently written. HTML allows users to produce web pages that include text, graphics and pointers to allow web pages.

HTML is a markup language, which means that formatting commands or tags written directly into the source file. Tags are interspersed with ordinary text and are not interspersed until the file is displayed or printed by a browser program. By embedding all the markup commands within each HTML file and standardized then, it becomes possible for any web browser to read and reformat any web page.

2.11 Point to Point Protocol (PPP):

Point to point in the Internet protocol environment a protocol for direct communication between two nodes over a serial point to point links such as between Routers in the Internet work or between a node and a router.

PPP is used to as a medium speed access protocol to the Internet. The protocol replaces the older SLIP (Serial Line Internet Protocol).

2.12 Virtual Private Networks (VPN):

Many companies have offered plans scattered over many cities sometimes over multiple companies. In the older days before public data networks it was common for such companies to leased lines from the telephone company between some or all pairs of locations. A network build-up from computers and leased telephone lines is called a Private Network.

Private networks were fine and very secure. But the costs involved with them are very high. The situation soon led to the invention of VPNs (virtual Private networks) which overlay networks on top of the public networks but with most of the properties of private networks.

They are called “virtual “because morally an illusion, just as virtual circuits is not real circuits. An increasingly popular approach is to build VPNS directly over Internet. A common design is to equip each office with a firewall and create tunnel to the Internet between all pairs of offices.

Which the system is brought up each pair of firewalls to negotiate the parameters of its SA (Secure Associations) including the servers, nodes,

algorithms and keys. Thus firewalls VPNS and IP sec with ESP in tunnel node are a natural communication and widely used in practice.

The key advantage of organizing a VPN this way is that it is completely transparent to all user software.

BASICS OF OOA & UML

3. BASICS OF OOA AND UML

3.1 Introduction:

This section describes the basics of Object Oriented Analysis (OOA) and Unified Modeling Language (UML) along with the usage of Rational Rose Tool.

3.2 Basic Concepts of OOPS:

3.2.1 Object

Objects are the basic run time entities representing a person or a place for a Bank account or a table of data in the real world. Objects take up space in the memory when a program is executed and they have an associated address. Objects interact by sending messages to one another each object contains data and code to manipulate the data. C++ is more emphasized on data rather than procedures (functions).

3.2.2 Class

The entire set of data and code of an object can be made user-defined data type with the help of class. A class defines a user-defined data type that is a collection of data and functions that manipulate the data together. Class provides a logical structure for the objects. Once a class is well defined we can create any number of objects of that type. Classes are user-defined data types and behave like the built-in types of a programming language.

3.2.3 Data Encapsulation

The wrapping up of data and methods into a single unit is called encapsulation. By using this concept of class we can bind the data and functions together to form a single unit that is a User Defined data Type (UDT). The main idea behind this is to provide security for the data that belongs to object, the data is not accessible to the outside world and only those member functions of the

class can access them. These functions provide interface between the object's data and the program thus the data can be insulated from direct access by program this is called data hiding.

3.2.4 Data Abstraction

Abstraction refers to the act of representing the essential features without including the background details or explanations. Classes give the concept of abstraction to abstract the data being accessed by non-member functions. Since classes use the concept of data abstraction they are known as abstract data types.

3.2.5 Inheritance

In OOP, the concept of inheritance provides the idea of reusability. It is a process by which objects of one class acquire the properties of objects of other class. It supports the concept of hierarchical classification. The main idea of inheritance is to extend the functionality of a class without destroying or redefining it. We can add additional features to a class without modifying it, this is possible by deriving a new class from an existing class. The new class will have combined features of both the classes.

3.2.6 Polymorphism

Polymorphism is the ability to take more than one form. Exhibiting more than one form by the same function or operator a several different contexts is called polymorphism and the function or the operator is said then overloaded. Polymorphism is the attribute that allows one interface to control access to a general class of actions. The specific action selected is determined by the exact nature of the situation. Thus polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. This concept is extensively used in implementing inheritance.

3.2.7 Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the runtime. It is associated with polymorphism and inheritance.

3.2.8 Message Communication

An OOP consists of a set of objects that communicate with each other hence object need to communicate with one another by sending and receiving the information much the same way as people pass messages to one another. The concept of message passing makes it easier to build systems that directly model the real world applications. A message for an object is a request for execution for a procedure and therefore will invoke a function in the receiving object that generates the desired result. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

3.3 Features of Objected Oriented Programming

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as Objects.
- Data structures are divided such that they characterize the Objects.
- Functions that operate on the data of an object are tied together in the structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data functions can easily be added whenever necessary.

3.4 Object – Oriented Programming Paradigm:

This major motivating factor in the invention of Object Oriented approach is to salvage some of the flaws encountered in the procedural approach. OOP treats data as critical elements in the program development and does not

allow it to flow freely around the system. It ties data more closely to the functions that operate on it and protects it from accidental modifications from outside functions. OOP's allows us to decompose a problem into number of entities called Objects and then builds data and functions around these entities. The data of an object can be accessed by the functions associated with the Object.

An Object Oriented programming language supports a special data type known as a Class that is similar to a structure in C but with a variation. A class contains a set of heterogeneous variables along with a set of functions to manipulate the given variables. The variables with in a class are known as data members and the functions are known as member functions. Every variable that is declared by the type of a Class is known as an Object where, every Object shares the same elements of the class.

Any programming language is said to be an Object Oriented programming language if it satisfies all the eight basic concepts of OOP's.

3.5 Unified Modeling Language:

Unified Modeling Language (UML), a standard modeling language is being used. When coupled with best practices of developing software in *iterative incremental model*, *visual modeling* helps in exposing and assesses architectural changes and communicates those changes to the entire development team.

3.6 Rational Rose Case Tool:

UML is supported by *Rational Rose Case tool* .It provides comprehensive integrated tools for requirements management, configuration management, drawing model diagrams, code generation, testing (unit as well as system), report and documents generation etc. It is the tool that supports any software development. The Rational Rose product family is designed to provide the software developer with a complete set of visual modeling tools for

development of robust, efficient solution to real business needs in the client server, distributed enterprise and real time system environments.

Tools provide capability to:

- Identify and design business objects and then map them to software.
- Partition services across a tiered service model.
- Design how components will be distributed across a network.
- Generate code framework directly from the model.
- Use reverse engineering to create models for existing components and applications.
- Use round trip engineering facility to keep the design synchronous with the code.

3.6.1 Use – Case Diagrams:

The behavior of the system under development, that is the functionality that must be provided by the system while documenting a use case model that illustrates the system's intended functions (use cases), its surroundings (actors), and relationships between the use cases and the actors (use case diagrams). The most important role of the use case model is that of a communication among the developers and the end users. It provides a vehicle used by the customers or end users and the developers to discuss the system's functionality and behavior.

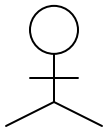
The use case model starts in the Inception phase with the identification of the actors and principal use case for the system. The model is then developed in the Elaboration phase where more detail information is added to the identified use cases, and additional use cases are added as on needed.

3.6.1.1 Actors

Actors are not part of any system; they represent any thing or any one that must interact with the system. An actor may

- Only input information to the system.
- Only receive information from the system.
- Input and receive information to and from the system.

Typically these actors are found in the problem statement and by conversations with customers and domain experts. In the UML, an actor is represented as a stickman.



UML Notation for an Actor

3.6.1.2 Use Cases

Use Cases model a dialogue between an actor and the system. They represent the functionality provided by the system i.e., what capabilities will be provided to an actor by the system. The collection of use cases for a system constitutes all the defined ways the system may be used.

3.6.1.2.1. The formal definition for a use case:

A use case is a sequence of transactions performed by a system that yields a measurable result of value for a particular actor.

UML Notation for use case

3.6.1.2.2 The Flow of events for a Use Case

Each use case is documented with a flow of events. The flow of events for a Use case is a description of the events required to accomplish the required

behavior of the use case. The flow of events is written in terms of what the system should do, not how the system does it.

3.6.1.2.3 Use Case Relationships

There are two types of relationships that may exist between Use cases:

- uses
- extends

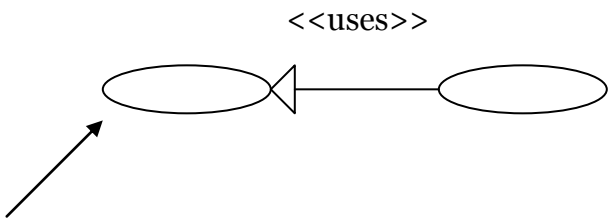
Multiple use cases may share pieces of the same functionality. This functionality is placed in a separate use case rather than documenting it in every use case that needs it. Uses relationships are created between the new use case and any other use case that uses its functionality. A use case relationship is drawn as arrow with a large hollow triangle (generalization arrow) at the end closest to the use case. An example of extends relationship is shown below

3.6.1.2.4 Optional behavior:

Its behavior is to run under certain conditions, such as triggering an alarm. Several different flows, which may be run, based on actor selections.

3.6.1.3 Use Case Diagram:

A Use Case diagram is a graphical view of some or all of the actors use case, and their interactions identified for a system, i.e., it shows the interaction between use cases, which represent system functionality and actors, which represent the people or system that provide or receive information from the system. Each system typically has a main use case diagram, which is a picture of a system boundary (actors) and the major functionality provided by the system (use case). Other use case diagrams may be created as needed.



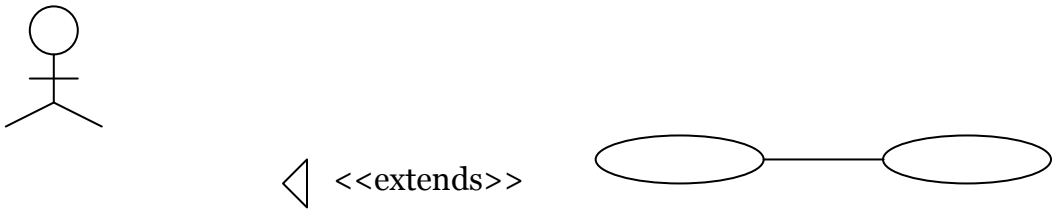


Fig: Use Case relationships

Note:

The remaining concepts of UML have been discussed in the Design section of the Report.

WIN 32

APPLICATION PROGRAMMING

INTERFACE

4. WIN 32 APPLICATION PROGRAMMING INTERFACE

4.1 Introduction:

This section gives an overview and discussion of the Windows 32 Application Programming Interface (API) which serves to develop comprehensive windows based programs that can run on all Windows Operating Systems.

4.2 Overview:

The Microsoft Win32 Application Programming Interface (API) allows applications to exploit the power of the 32-bit Windows family of operating systems. Using the Win32 API, you can develop applications that run successfully on all 32-bit versions of Windows while taking advantage of the features and capabilities unique to each version.

Differences in the implementation of the programming elements depend on the capabilities of the underlying features of the platform. The most notable difference is that some functions are supported only on the more powerful platforms.

The Win32 API can be used in all Windows-based applications.

4.3. Functional Categories of Win32 API

The Win32 API consists of the following functional categories:

4.3.1 Base Services:

The base services functions give applications access to the resources of the computer and the features of the underlying operating system, such as memory, file systems, devices, processes, and threads. An application uses these functions to manage and monitor the resources it needs to complete its work. For example, an application uses memory management functions to allocate and free memory. Process management and synchronization functions start and coordinate the operation of multiple applications or multiple threads of execution within a single application.

The file I/O functions provide access to files, directories, and input and output (I/O) devices. These functions give applications access to files and directories on disks and other the storage devices on a specified computer and on computers in a network. The file I/O functions support a variety of file systems, including the FAT file system, the CD-ROM file system (CDFS), and NTFS.

Applications can share code or information with other applications. For example, they can make useful procedures available to all applications by placing

these procedures in DLLs. Applications access these procedures by using DLL functions to load the libraries and retrieve the addresses of the procedures. Communications functions read from and write to communications ports as well as control the operating modes of these ports. For interprocess communication (IPC), there are DDE, pipe, mailslot, and file mapping functions.

Registry and initialization functions let applications store application-specific information in system files so that new instances of the application or even other applications can retrieve and use the information.

Applications handle special conditions during execution. For example, they can handle errors, log events, and handle exceptions. Applications can also use special functions to debug code and improve its performance. For example, the debugging functions permit single-step control of the execution of other processes, and the performance monitoring functions provide detailed information on process execution.

4.3.2 Common Control Library:

The shell incorporates a number of controls that help give Windows its distinctive look and feel. Because these controls are supported by DLLs that are a part of the operating system, they are available to all applications. Using the common controls helps keep an application's user interface consistent with that of the shell and other applications. Because developing a control can be a substantial undertaking, using the common controls can also save you a significant amount of development time.

The common controls are a set of control windows supported by the common control library Comctl32.dll. Like other controls, a common control is a child window that an application uses in conjunction with another window to perform I/O tasks. The common control DLL includes a programming interface that applications use to create and manipulate the controls as well as to receive user input.

4.3.3 Graphics Device Interface:

The graphics device interface (GDI) provides functions and related structures that an application can use to generate graphical output for displays, printers, and other devices. Using GDI functions, you can draw lines, curves, closed figures, paths, text, and bitmap images. The color and style of the items you draw depends on the drawing objects — that is, pens, brushes, and fonts — that you create. You can use pens to draw lines and curves, brushes to fill the interiors of closed figures, and fonts to write text.

Applications direct output to a specified device by creating a device context (DC) for the device. The device context is a GDI-managed structure containing

information about the device, such as its operating modes and current selections. An application creates a DC by using device context functions. GDI returns a device context handle, which is used in subsequent calls to identify the device. For example, using the handle, an application can retrieve information about the capabilities of the device, such as its technology type (display, printer, or other device) and the dimensions and resolution of the display surface.

Applications can direct output to a physical device, such as a display or printer, or to a "logical" device, such as a memory device or metafile. Logical devices give applications the means to store output in a form that is easy to send subsequently to a physical device. After an application records output in a metafile, it can play that metafile any number of times, sending the output to any number of physical devices.

Applications use attribute functions to set the operating modes and current selections for the device. The operating modes include the text and background colors, the mixing mode (also called the binary raster operation) that specifies how colors in a pen or brush combine with colors already on the display surface, and the mapping mode that specifies how GDI maps the coordinates used by the application to the coordinate system of the device. The current selections identify which drawing objects are used when drawing output.

4.3.4 Network Services:

The network functions allow communication between applications on different computers on a network. You can use these functions to create and manage connections to shared resources, such as directories and network printers, on computers in the network.

The network interfaces include Windows Networking and Network Management. Windows 95/98 supports a subset of these functions. To determine whether a function is supported on Windows 95/98, see the Requirements section in the documentation for the function.

4.3.5 User Interface:

The user interface functions give applications the means to create and manage a user interface. You use these functions to create and use windows to display output, prompt for user input, and carry out the other tasks necessary to support interaction with the user. Most applications create at least one window.

Applications define the general behavior and appearance of their windows by creating window classes and corresponding window procedures. A window class identifies default characteristics such as whether the window processes mouse button clicks or has a menu. The corresponding window procedure

contains code that defines the behavior of the window, carries out requested tasks, and processes user input.

Applications generate output for a window by using the GDI functions. Because all windows share the display screen, applications do not receive access to the entire screen. Instead, the system aligns and clips all output to fit within the corresponding window. Applications can draw in a window in response to a request from the system or while processing input messages. When the size or position of a window changes, the system typically sends a message to the application requesting that it paint any previously unexposed area of its window.

Applications receive mouse and keyboard input in the form of messages. The system translates mouse movement, mouse button clicks, and keystrokes into input messages and places these messages in the message queue for the application. The system automatically provides a queue for each application. The application uses message functions to extract messages from the queue and dispatch them to the appropriate window procedure for processing.

Applications can process the mouse and keyboard input directly or let the system translate this low-level input into command messages by using menus and keyboard accelerators. You use menus to present a list of commands to the user. The system manages all actions required to let the user choose a command and then sends a message to the window identifying the choice. Keyboard accelerators are application-defined combinations of keystrokes that the system translates into messages. Accelerators typically correspond to commands in a menu and generate the same messages.

Applications often respond to command messages by prompting the user for additional information with dialog boxes. A dialog box is a temporary window that displays information or requests input. A dialog box typically includes controls — small, single-purpose windows — that represent buttons and boxes through which the user provides information. There are controls for typing text, scrolling text, selecting items from a list of items, and so on. Dialog boxes manage and process the input from these controls, making this information available to the application so that it can complete the requested command.

You can share useful data, such as bitmaps, icons, fonts, and strings, by adding this data as "resources" to the file for an application or DLL. Applications retrieve the data by using the resource functions to locate the resources and load them into memory.

Window management functions provide other features related to windows, such as carets, the clipboard, cursors, hooks, icons, and menus.

4.3.6 Windows Shell:

Your applications can use the shell interfaces and functions to enhance various aspects of the Windows shell.

A namespace is a collection of symbols, such as file and directory names or database keys. The shell uses a single hierarchical namespace to organize all objects of interest to the user, including files, storage devices, printers, and network resources. The namespace is similar to the directory structure of a file system, except that the namespace contains objects other than files and directories.

A shortcut (also called a shell link) is a data object that contains information used to access another object located anywhere in the shell's namespace. A shortcut allows an application to access an object without having to know the current name and location of the object. Objects that are accessible through shortcuts include files, folders, disk drives, printers, and network resources.

There are several ways to extend the shell. The system uses icons to represent files in the shell's namespace. By default, the system displays the same icon for all files that have the same file name extension. An icon handler can override the default and set the icon for a particular file. A context menu handler is a shell extension that modifies the contents of a context menu. The system displays a context menu when the user clicks or drags an object using mouse button 2. The context menu contains commands that apply specifically to the object that was clicked or dragged. Most context menus have a **Properties** command that displays the property sheet for the selected item. A property sheet contains information about an object in a set of overlapping windows called pages. A property sheet handler is a shell extension that adds pages to a system-defined property sheet or replaces pages in a Control Panel application's property sheet. A copy hook handler is a shell extension that approves or disapproves the moving, copying, deleting, or renaming of a file object.

The shell includes a **Quick View** command that allows the user to view the contents of a file without having to run the application that created it. A file viewer provides a user interface for viewing a file. The shell uses the file name extension to determine which viewer to run. You can provide file viewers for new file formats or replace an existing viewer with one that includes more functionality. A file viewer works in conjunction with a file parser, which provides the parsing needed to generate the Quick View of a file of a specified type. You can provide additional file parsers to support new file types.

SOFTWARE REQUIREMENT'S SPECIFICATION

5. SOFTWARE REQUIREMENT'S SPECIFICATION

5.1 Introduction

5.1.1 Purpose**:**

The purpose of this SRS document is to describe the comprehensive requirements for the proposed "Remote Monitoring System".

5.1.2 Scope**:**

This document is the only one that describes the requirements of the system. It is meant to be used as a basis for validating the final delivered system.

5.1.3 Definitions, Acronyms, Abbreviations:

R.M.S: Remote Monitoring System.
G.U.I: Graphical User Interface.
U.M.L*:* Unified Modeling Language.

5.2 General Description

5.2.1 Product perspective:

It is proposed to take up Personal Computer Remote monitoring in the Network Environment, for Monitoring & Controlling various Nodes/Servers connected in the Clusters.

This Project will help Users/Supervisors to" Monitor" the activities assigned to the individual operator/programmer at the Clients.

5.2.2 Product Functions:

The Remote Monitoring System, when activated by the user opens up with a user friendly Graphical User Interface with adequate features to facilitate effective usage of the system.

The Remote Monitoring System runs (operates) on the Server Machine of the Network and Monitors all the Clients connected to the Server.

The following are the functions of RMS:

(a) Display of Computer Name, IP address of the Clients.

(b) Display of the User currently logged on in the Clients.

(c) Display of the Desktop Image of the Client Systems.

In addition to the above the following functions are also provided by the RMS.

Control operations can be performed on the selected Client such as:

i. Logoff.

ii. Re Start.

iii. Shut Down.

5.2.3 General Constraints:

The main constraint on the Remote Monitoring System is that the RMS can be used for monitoring only those users of the client machines who have logged on after the RMS started running on the Server i.e. it cannot monitor those users who have logged on the client machines before RMS was started on the server.

The Remote Monitoring System is conceived to work in the Windows based environments only.

5.2.4 Software Requirements:

Server Side:

Operating Systems: Windows NT 4.0 (or) Windows 2000 Servers.

Client Side:

Operating Systems: Windows NT 4.0 Workstation, Windows 98, Windows Me, Windows XP e.t.c.

5.2.5 Hardware Requirements:

Server Side:

Processor: Pentium series or higher (or equivalent).
RAM: 128 MB or above.

Client Side:

Processor: Pentium series or higher (or equivalent).
RAM: 64MB or as per OS.

5.2.6 Network:

The Network, LAN (Ethernet) should be available so as to allow the communication between the Server and Clients. A typical Client–Server network architecture is required by the RMS.

5.3 Specific Requirements:

5.3.1 Functional Requirements:

The Use-Case analysis of the Functional specifications is discussed here.

Six Use-Cases were identified which completely describe the System and were analyzed with the help of Rational Rose Tool for the Unified Modeling Language.

They are

1. Displaying the Client Desktop Image and Zooming it to Required Level.
2. Getting System Information of the Client.
3. Performing Control Functions on the Client.
4. Sending Messages to the Client.
5. Sending the Client Information to the RMS.
6. Getting the Desktop of the Client.

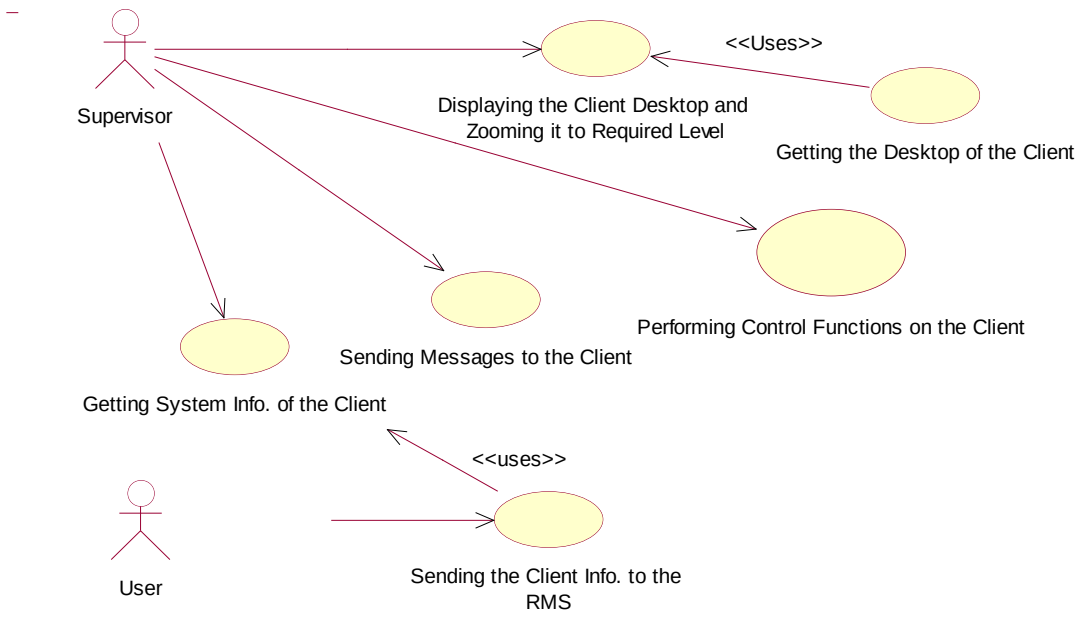
There are two basic *actors* interacting with the system

- Supervisor: The operator who uses RMS at the server and *monitors* the Clients.
- User: The user who is currently *logged on* at a Client system.

The Use – Case 1 (Displaying the Client Desktop Image and Zooming it to required level) uses the Use – Case 6 (Getting the Desktop from the Client).

The Use – Case 2 (Getting System Information of the Client) uses the Use – Case 6 (Sending the Client Information to the RMS).

The Use-Case Diagram for the RMS is:



5.3.1.1 Use-Case 1:

Name: Displaying the Client Desktop Image and Zooming it to Required Level.

Brief Description

This use case enables the Supervisor to observe the Desktop Image of a Client at his selects zoom level.

Flow of Events

Basic Flow:

The Supervisor initiates this Use-Case by selecting a Client system in the GUI based Front-End of the RMS.

1. Supervisor selects a Client system.
2. Supervisor requests for the Desktop Image of the Client
3. RMS requests the Client for its Desktop Image.
4. Client system sends its Desktop Image.
5. RMS prompts Supervisor for Required Zoom Level.
6. Supervisor selects the required level.
7. RMS Zooms and Displays the Desktop Image as requested.

Alternate Flow:

If the selected Client is not active then the system displays error message.

Pre Conditions:

The Supervisor has to select a Client system.

Post Conditions:

Selected Client Desktop image is displayed at selected zoom level.

5.3.1.2 Use-Case 2:

Name: Getting System Information of the Client.

Brief Description

This use case enables the Supervisor to get the system information of a Client.

Flow of Events

Basic Flow:

The Supervisor initiates this Use-Case by selecting a Client system in the GUI based Front-End of the RMS and requesting for Client system Information.

1. Supervisor selects a Client system.
2. Supervisor requests for Client System Information from the Menu
3. RMS displays the System Information as requested.

Alternate Flow:

If the selected Client is not active then the system displays error message.

Pre Conditions:

The Supervisor has to select a Client system.

Post Conditions:

Selected Client system information is displayed by the RMS.

5.3.1.3 Use-Case 3:

Name: Getting Performing Control Functions on the Client.

Brief Description

This use case enables the Supervisor to perform *Control* operations on a Client.

Flow of Events
Basic Flow:

The Supervisor initiates this Use-Case by selecting a Client system in the GUI based Front-End of the RMS and requesting for Control Menu.

1. Supervisor selects a Client system.
2. Supervisor selects desired Control operation from the Control Menu
3. RMS prompts for Message to be displayed at the Client.
4. Supervisor enters the Message.
5. Displays the Message at Client and performs the Control operation.

Alternate Flow:

If the selected Client is not active then the system displays error message.

Pre Conditions:

The Supervisor has to select a Client system.

Post Conditions:

Selected Control operation is performed on the selected Client.

5.3.1.4 Use-Case 4:

Name: Sending Messages to the Client.

Brief Description

This use case enables the Supervisor to send Messages to the Client.

Flow of Events

Basic Flow:

The Supervisor initiates this Use-Case by selecting a Messages Menu in the GUI based Front-End of the RMS.

1. Supervisor selects Send Message option from the Menu.
2. RMS prompt for Message and Clients to which the Message must be sent
3. Supervisor enters the Message and selects the Client.
4. RMS sends the Message to the Client.

Alternate Flow:

If the selected Client is not active then the system displays error message.

Pre Conditions:

The Supervisor has to select Message from the Menu.

Post Conditions:

RMS sends the Message to the Selected Client.

5.3.1.5 Use-Case 5:

Name: Sending the Client Information to the RMS.

Brief Description

This use case enables the Client to send its system information to the RMS.

Flow of Events

Basic Flow:

The User initiates this Use-Case when he log's on the Client System.

1. User Log's on the Client.
2. Client collects the System Information.

3. Client sends the Collected System Information to the RMS at Server.
4. RMS stores the received Information.

Pre Conditions:

The User has to Log on to the Client to start this Use-Case.

Post Conditions:

RMS gets the system information of the Client.

5.3.1.6 Use-Case 6:

Name: Getting the Desktop of the Client.

Brief Description

This use case enables the RMS to regularly update the Desktop Image of the Client.

Note

This is an *abstract* use case and is used by RMS to regularly update the Desktop Image of the Client.

Flow of Events

Basic Flow:

A Timer initiates this Use-Case at predefined intervals.

1. RMS requests for the Client's Desktop Image.
2. Client collects and sends the Desktop Image to RMS.
3. RMS stores and Displays the Desktop Image of the Client.

Alternate Flow:

If the selected Client is not active then the system displays error message.

Pre Conditions:

The Timer has to start this Use-Case.

Post Conditions:

RMS gets the updated Desktop Image of the Client.

5.3.2 External Interface Requirements:

5.3.2.1 User Interface:

The User Interfaces are GUI based, simple to understand and to use, Windows developed in Visual Studio 6.0.

5.3.2.2 Communication Interface:

The Network Interface Card (NIC) is required as an Interface to the LAN.

5.3.3 Performance Constraints:

The Desktop Images of the Clients Connected to the Server have to be true for 90% of Time (i.e. Updating has to be done quite regularly).

5.3.4 Design Constraints:

(1) The modification or actions such as

- A System getting disconnected or
- A new System getting connected to the LAN.
- A User Log Off or
- A User Log On

Have to be currently traced 100% of the time (i.e. every such action has to be Traced).

(2) The Transfer of data especially the Desktop Images should not exceed 10% of the Bandwidth of the network.

(3) The preparation of Image to be sent across with all the Communication Overhead Should not take more than 10% of the processor time at the Client Machines.

(4) The receiving of the Image at the Server side and sending the Image and its display in the RMS should not take more than 10% of the Processor's time.

SOFTWARE DESIGN DOCUMENT

6. SOFTWARE DESIGN DOCUMENT

6.1 Introduction

6.1.1 Purpose:

The purpose of this document is to describe and discuss the development strategy, Architecture and Design of the *"Remote Monitoring System"*.

6.1.2 Scope:

This document describes the design of the *"Remote Monitoring System"* based on the Rapid Application Development paradigm of the Software Engineering.

6.1.3 Definitions and Acronyms:

R.M S: Remote Monitoring System
R.A.D: Rapid Application Development
U.M.L: Unified Modeling lan*g*uage

6.2 Rapid Application Development (RAD):

This is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a high-speed adaptation of the linear sequential model in which rapid development is achieved by using the component- based construction.

6.3 System Architecture:

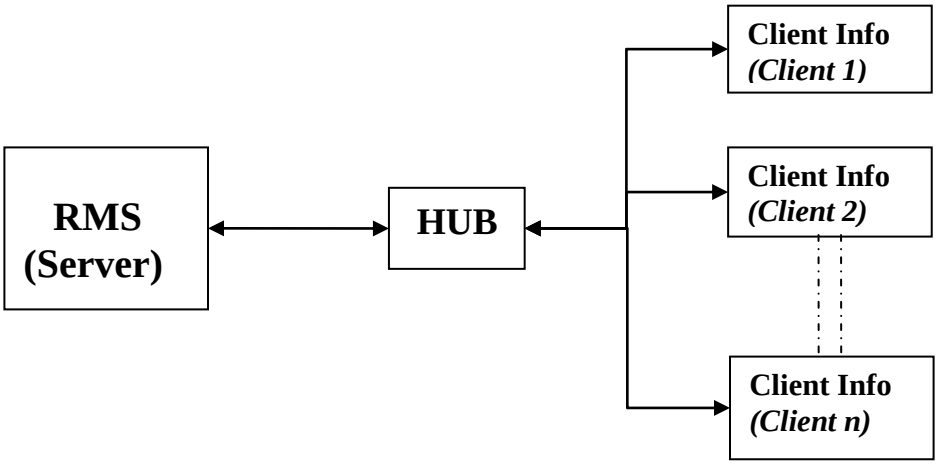
A well – designed architecture is the foundation for an extensible and flexible system. The architecture of the Remote Monitoring System has been designed in view of the above mentioned point.

The Remote Monitoring System follows the conventional 2–tier architecture. RMS is based on the Client – Server Network Architecture.

The RMS has two components:

i. Monitoring Component
ii. Client Info Component

The Architecture of the RMS is:



Note: The RMS shown in the server is the Monitoring Component

6.3.1 Monitoring Component:

The Monitoring component runs at the server and acts as an interface to the Supervisor, who uses RMS to monitor the Clients. The Monitoring component provides both the GUI based User Interface and Control features of the RMS.

The Monitoring component *depends* on the Client Info Component for necessary information from the Client systems.

6.3.2 Client Info Component:

This component resides at the Client machines. Its main purpose is to collect necessary information and sends it to the Monitoring Component as per the need and demand of the situation.

6.4 System Design:

The design of the system may be defined as "the process of applying various techniques and principles for the purpose of defining advice, a process or a system in sufficient detail to permit its physical realization…"

Our main goal is to produce a model representation of an entity that will later be built. We have used the strategy of Rapid Application Development paradigm to develop the system and Unified Modeling Language to design the system. The Rational Rose tool has been used as a visual modeling tool.

6.4.1 Sequence Diagrams:

Sequence diagrams are interaction diagrams that are ordered by time, we read them from top to bottom. They illustrate how objects interact with each other. They focus on message sequences, that is, how messages are sent and received between number of objects. Sequence diagrams have two axes: the vertical axis shows time and the horizontal axis shows a set of objects. A sequence diagram also reveals the interaction for a specific scenario – a specific interaction between the objects that happens at some point in time during the system's execution (e.g., when a specific function is used).

In the context of RMS we have two Classes:

i. RMS: This Class represents the Monitoring Component
ii. Client: This Class represents the Client Info component, as discussed in the Architecture respectively.

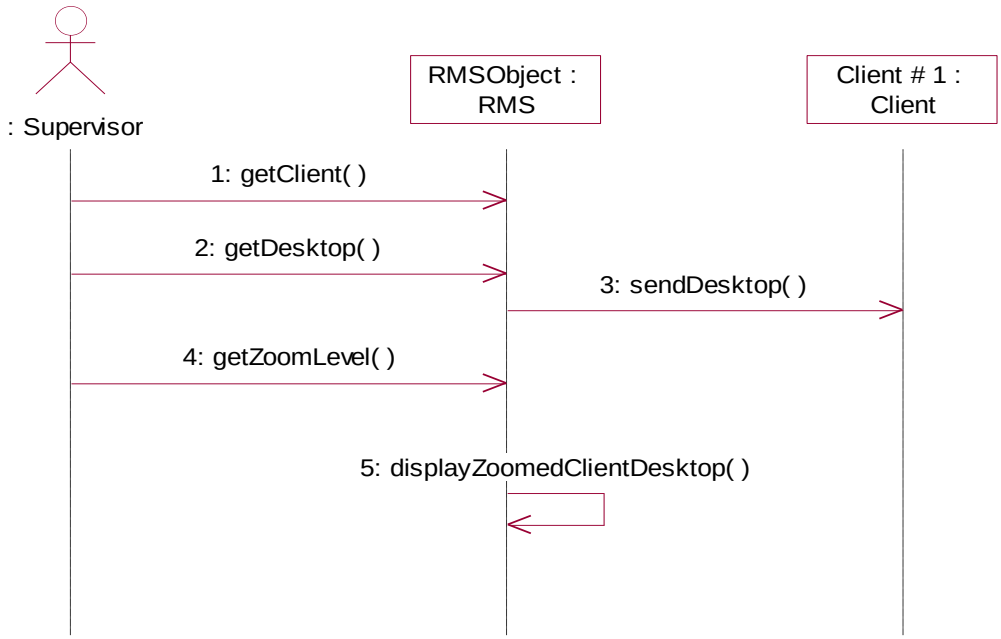
The Objects RMS Object and Client #1 are used to represent the RMS and Client Classes respectively.

Supervisor and User are the actors concerned with the RMS.
Supervisor: Supervisor is the one who operates RMS at the Server.
User: User is the user logged on to the Client System.

6.4.1.1 Scenario 1:

Name: Supervisor requests for the desktop of a Client #1

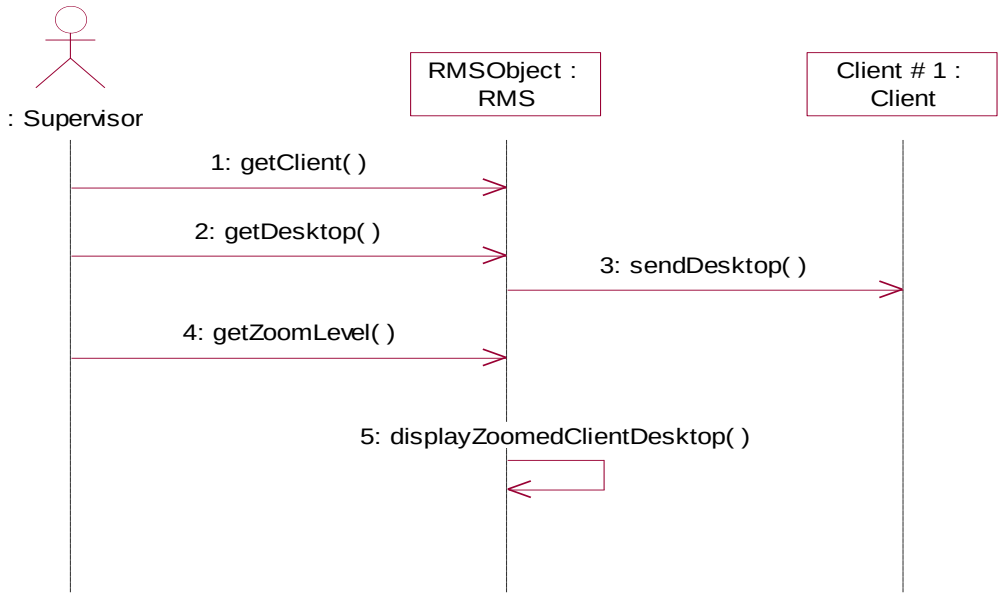
Sequence Diagram:



6.4.1.2 Scenario 2:

Name: Supervisor request Client #1 System's information

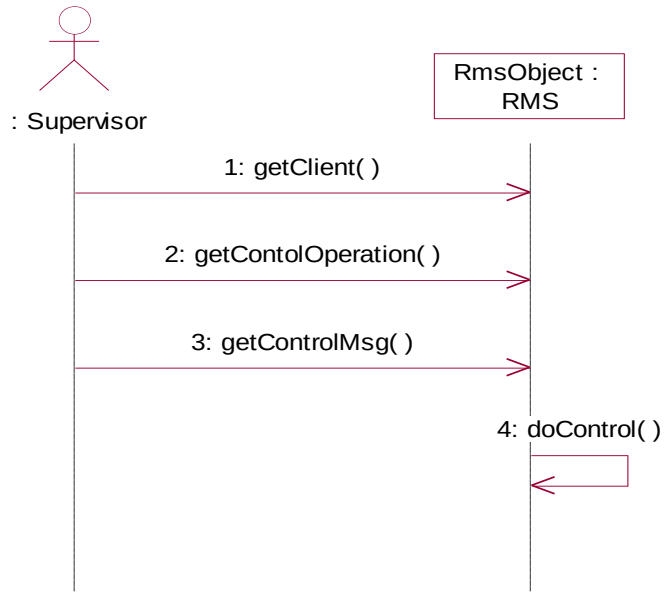
Sequence Diagram:



6.4.1.3 Scenario 3:

Name: Supervisor performs a Control operation on the Client #1

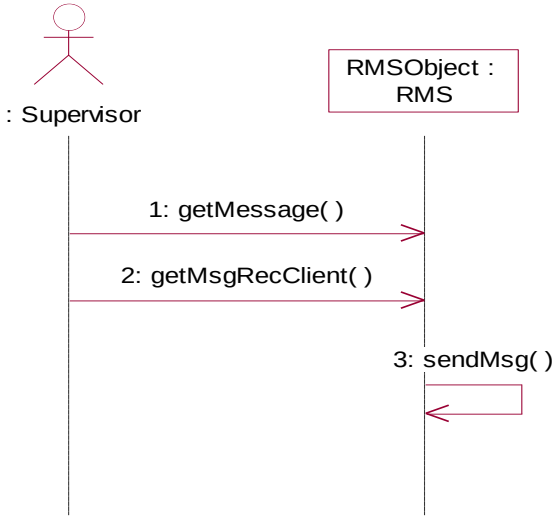
Sequence Diagram:



6.4.1.4 Scenario 4:

Name: Supervisor sends a message to User on Client #1

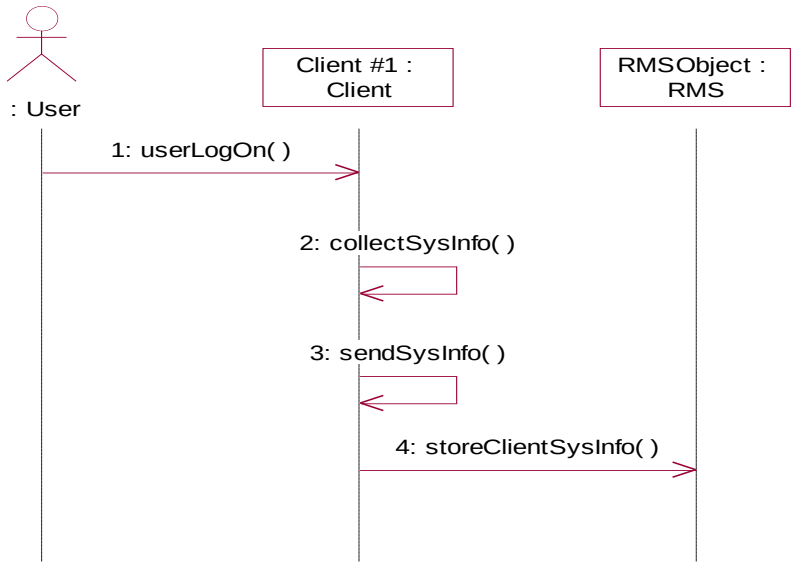
Sequence Diagram:



6.4.1.5 Scenario 5:

Name: Client #1 sends System Info. to the RmsObject.

Sequence Diagram:

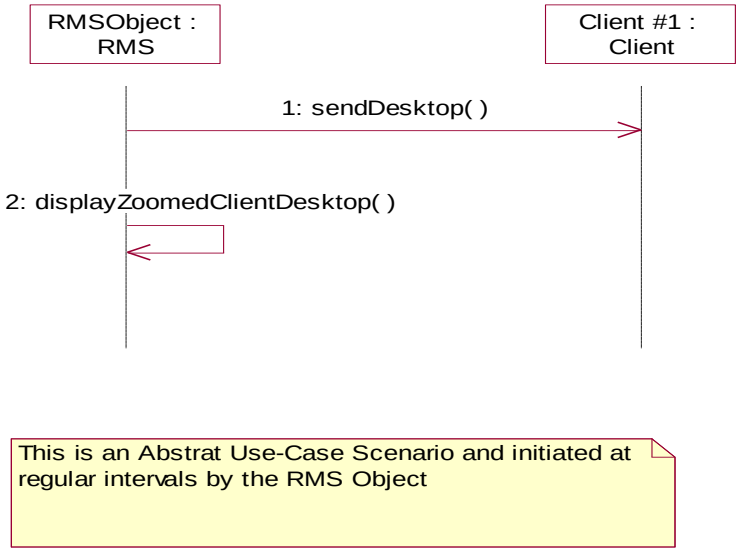


User is the actor who is logged on to the Client System.

6.4.1.6 Scenario 6:

Name: RMSObject request's for Client #1 Desktop Image

Sequence Diagram:



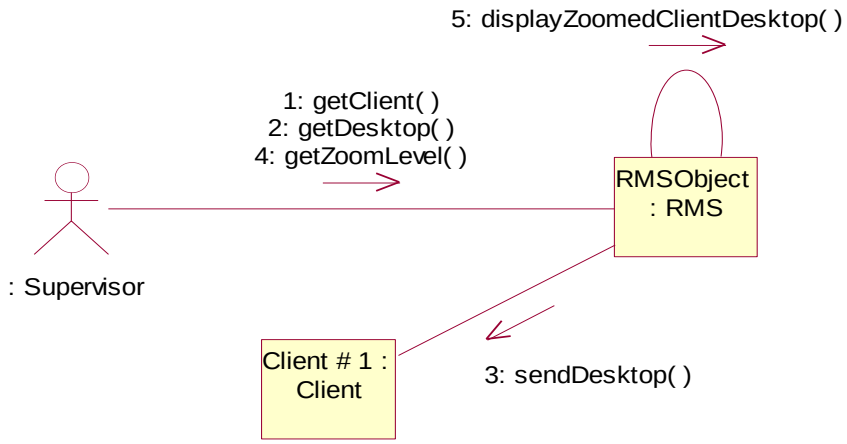
6.4.2 Collaboration Diagrams:

The Collaboration Diagrams are used to represent the same information as the Sequence Diagrams but focus more on the relationships between the Objects.

6.4.2.1 Scenario 1:

Name: Supervisor requests for the desktop of a Client #1

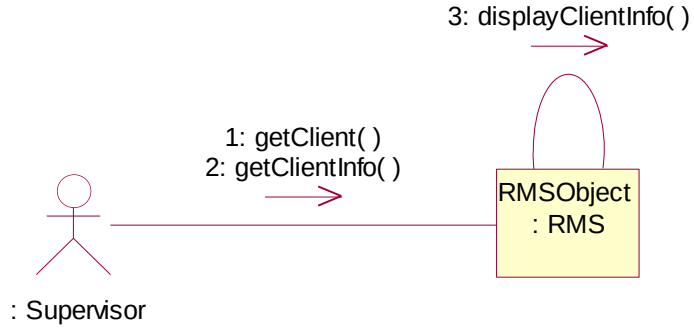
Collaboration Diagram:



6.4.2.2 Scenario 2:

Name: Supervisor request Client #1 System's information

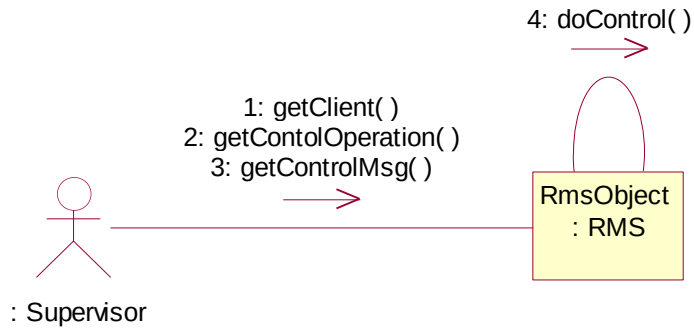
Collaboration Diagram:



6.4.2.3 Scenario 3:

Name: Supervisor performs a Control operation on the Client #1

Collaboration Diagram:



6.4.2.4 Scenario 4:

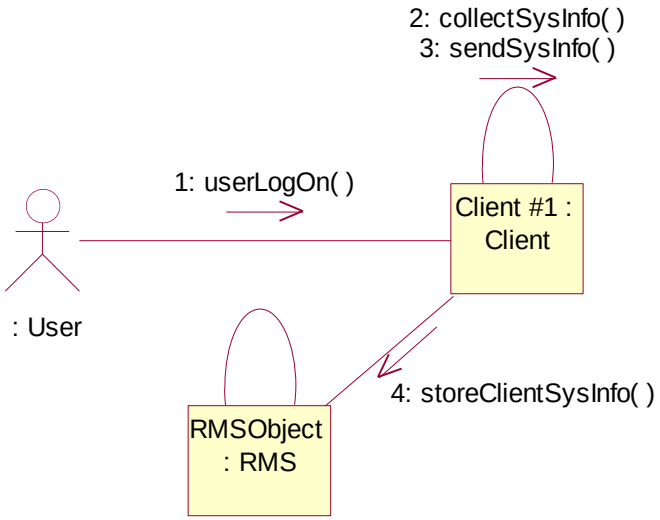
Name: Supervisor sends a message to User on Client #1

Collaboration Diagram:

6.4.2.5 Scenario 5:

Name: Client #1 sends System Information to the RmsObject.

Collaboration Diagram:

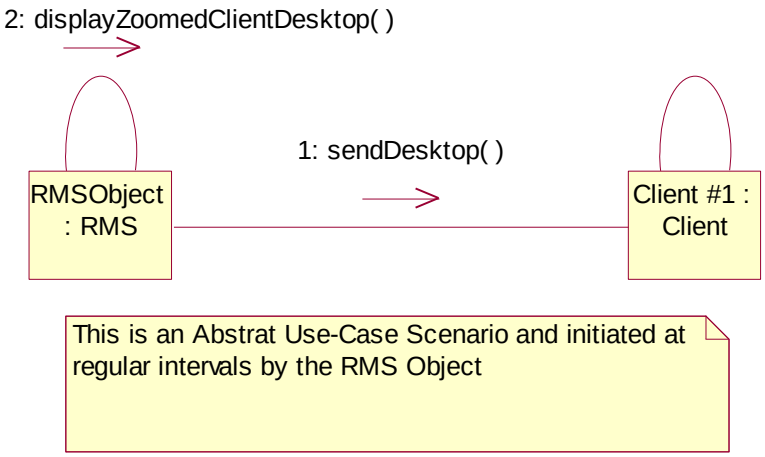


User is the actor who is logged on to the Client System.

6.4.2.6 Scenario 6:

Name: RMSObject request's for Client #1 Desktop Image

Collaboration Diagram:



6.4.3 Class Diagrams:

Class

A class is a description of a group of objects with common properties (attributes), common behavior (operations), common relationships to other objects (association and aggregation), and common semantics. A class is drawn as a compartmentalize rectangle. The compartments show the class name, its structure, and its behavior.

Class name
Attributes
Operations

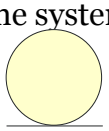
UML Notation for a Class

Class Diagrams

Class diagrams are created to provide a picture or a view of some or all of the classes in the model. Class diagrams may also be created in the use case view of model. These diagrams are typically attached to Use case and contain a view of the classes participating in the use case. The Main class diagram in the logical view of the model is typically a picture of the packages in the system. Each package also has its own Main class diagram, which typically displays the public class of the package. Other diagrams are created as needed.

Entity Class:

They are classes that are needed by the system to accomplish some responsibility. This type of class may reflect a real world Entity or it may be needed to perform tasks internal to the system.

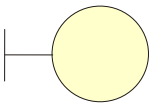


Entity Class

UML Notation for Entity Class

Boundary Class:

They handle the communication between the system surroundings and the inside of the system. They can provide the interface to a user or another system (Actor). They are also added to facilitate communication with other system.

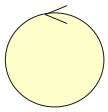


Boundary Class

UML Notation for Boundary Class

Control Class:

Control class model sequencing behavior specific to one or more Use Cases. They coordinate the events needed to realize the behavior specified on the Use Case. The Control Class can be thought as running or executing the Use Case. They represent the dynamics of the Use Case. Control Class typically is application dependent classes.



Control Class

UML Notation for Control Class

6.4.3.1 Class Diagrams for RMS:

As mentioned earlier there are two classes in the RMS. They are:

i. RMS Class:
ii. Client Class

6.4.3.1.1 RMS Class:

The RMS Class acts as both the boundary and control class. The RMS class is designed to run at the Server. It acts as boundary class by proving Interface to the Supervisor. It also passes the control instructions to the Client Class for transmitting and receiving required Information.

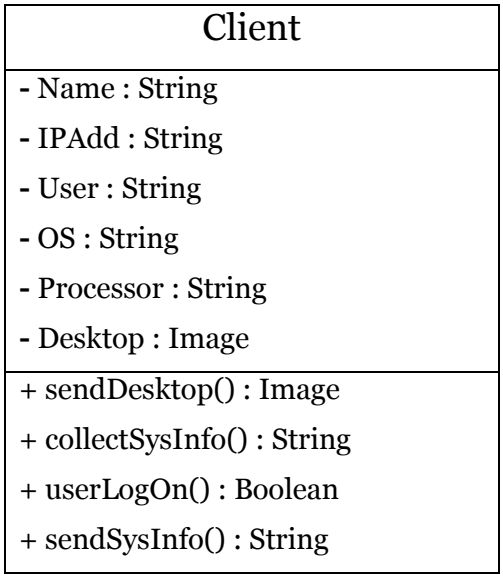
The RMS Class in UML Notation is:

RMS
- ServerName : String - IPAdd : String - No_of_Clients : Integer - Desktop_Images : Images - Clients : Strings - Clients_Info : Strings
+ getDesktop(Client_Name : String = Null) + getZoomLevel() : Integer + displayZoomedClientDesktop(ClientName : String, ZoomLevel : Integer) : Image + getClientInfo(ClientName : String) : String + getClient() : String + displayClientInfo() : String + getContolOperation(ClientName : String, Control : Integer = 0) + getControlMsg() : String + doControl(ClientName : String, Operation : Integer = 0, Message : String) : Boolean + getMsgRecClient() : String + getMessage() : String + sendMsg(ClientName : String, Message : String) : Integer + storeClientSysInfo(ClientName : String, Info : String)

6.4.3.1.2 Client Class:

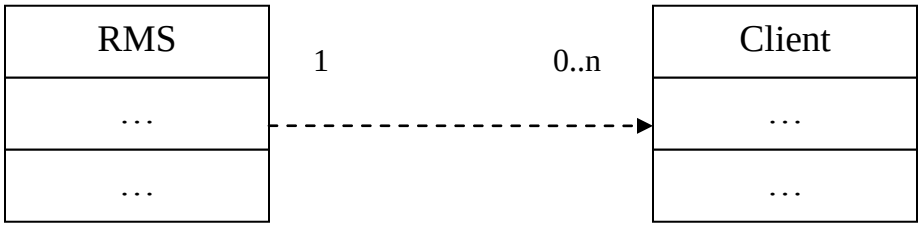
The Client Class stores the information about the Client systems and functions to send and receive information from the server. The Client is designed to collect the required information about the Client systems at the event of log on by a User. The class collects the information and sends it to the RMS Class present at the Server.

The Client Class in UML Notation is:



6.4.3.1.3 Dependency Relation:

There exists a dependency relation between the RMS and the Client as shown:



Note:

- The RMS Class acts as both the Control and Boundary Class.
- The Client Class represents the Client System.

6.4.4 Activity Diagrams:

Activity Diagrams represent the dynamics of the system. They are flow charts that are used to show the workflow of a system i.e. they show the flow of control from one activity to other activity in a system. Activity diagrams may be

created to represent the flow across use cases or they may be created to represent the flow within a particular use case.

Activity diagrams contain activities, transitions between the activities, decision points and synchronization bars. In the UML activities are represented as rectangles with round edges, transition are drawn as directed arrows, decision points are shown as diamonds and synchronization bars are drawn as thick horizontal or vertical bars.

6.4.4.1 Activity:

An activity represents the performance of some behavior in the workflow.

6.4.4.2 Transitions:

Transitions are used to shown the passing of the flow of control from activity to activity. They are typically triggered by the completion of the behavior in the originating activity.

6.4.4.3 Decision Points:

When modeling the workflow of a system it is often necessary to show where the flow of control branches based on a decision point. The transitions from a decision point contain a guard condition, which is used to determine which path from the decision point is taken. Decisions along with their guard conditions allow you to show alternate paths through a workflow.

6.4.4.4 Synchronization bars:

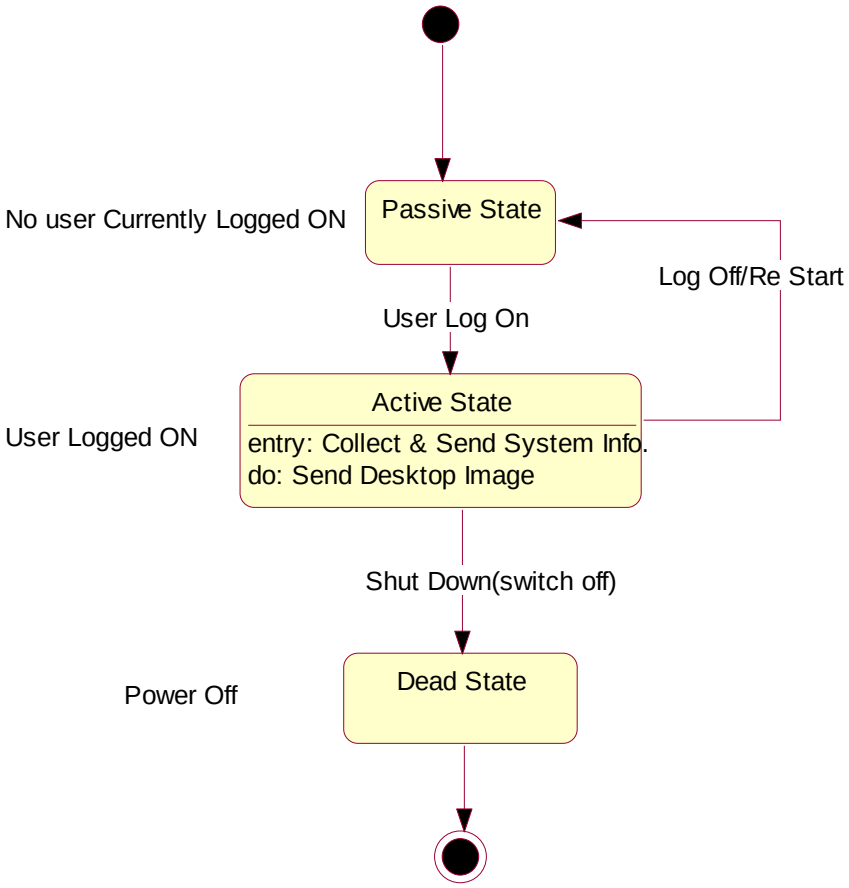
In a workflow there are typically some activities that may be done in parallel. A synchronization bar allows you to specify what activities may be done concurrently. Synchronization bars are also used to show joins in the workflow i.e. what activities must complete before processing may continue. A synchronization bar may have many incoming transitions and one outgoing transition, or one incoming transition and many outgoing transitions.

6.4.4.5 Initial and Final Activities

There are special symbols that are used to show the starting and final activities in a workflow. The starting activity is shown using a solid filled circle and the final activities are shown using a bull's eye. Typically, there is one starting activity for the workflow and there may be more than one ending activity.

6.4.4.6 Activity Diagram for Client Class:

In the context of RMS only Client Class has dynamic behavior as depicted



6.4.5 Component Diagrams:

Component diagrams show the physical view of a model. It shows the software components in a system and the relationship between them.

There are two types of components:

- Executable Libraries.
- Code Libraries.

6.4.5.1 Component Diagram for RMS:

RMS_EXE is an executable running on the Server.

ClientSide_Dll is a Dll residing at the Client.

6.4.6 Deployment Diagrams:

The deployment view is concerned with the physical deployment of the Application. This includes issues such as network layout and the location of components on the Network.

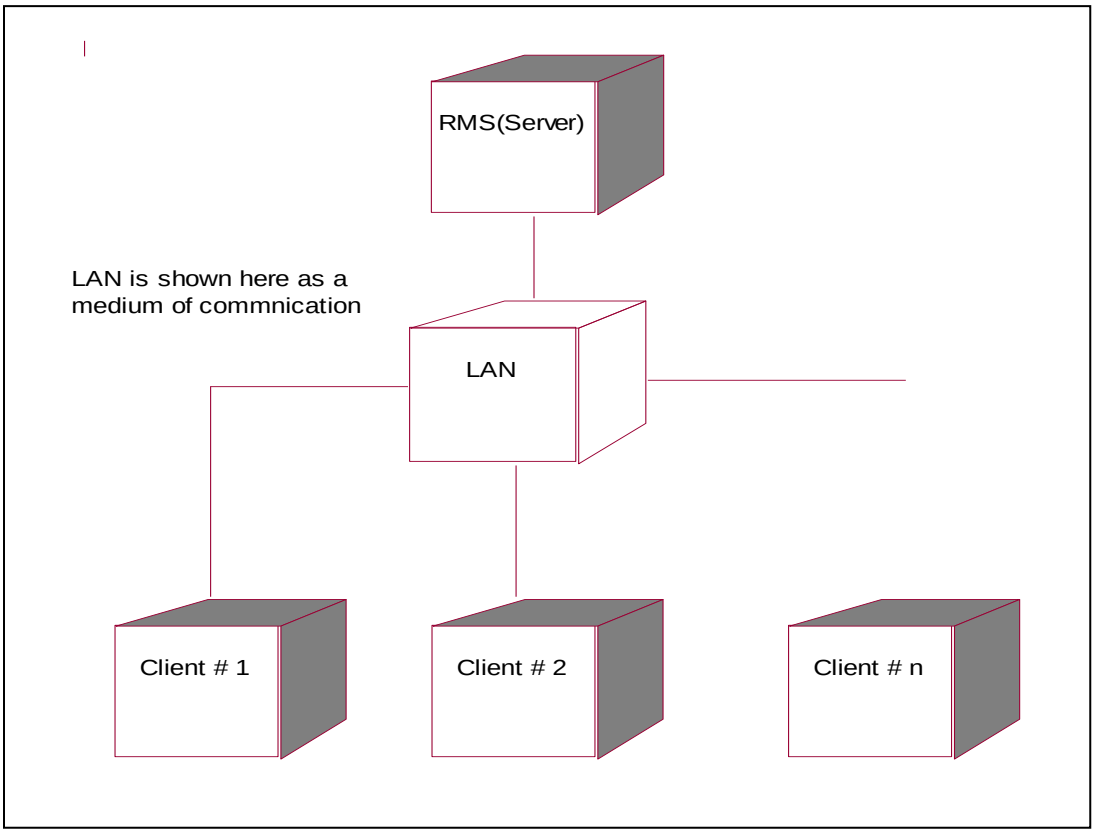
The Deployment view contains

- Processors,
- Devices,
- Process and
- Connections between processors and devices.

All of these are diagrammed on a Deployment Diagram.

There is only one deployment Diagram per system.

6.4.6.1 Deployment Diagram for RMS:



PSUEDO CODE

7. PSUEDO CODE

7.1 RMS Class:

7.1.1 Function Name: getDesktop(Client_Name : String = Null) : Image

Description : This function is used to get the Desktop Image of the Client system whose name is passed as a parameter.

Input: Client Name

Output: Desktop Image.

7.1.2 Function Name: getZoomLevel (): Integer

Description: This function gets the zoom level selected by the supervisor to display selected client desktop image.

Input: void.

Output: Integer

7.1.3 Function Name: displayZoomedClientDesktop(ClientName : String, ZoomLevel :
Integer) : Image

Description: This function is used to display the requested client desktop image to the required zoom level.

Input: Client system Name and the required zoom level

Output: The required Image.

7.1.4 Function Name: getClientInfo(ClientName : String) : String

Description: This function is used to get the Information about the Clients connected.

Input: Client system Name

Output: Information as a String.

7.1.5 Function Name: getClient() : String

Description: This function is used to get the Client system selected by the user.

Input: void

Output: Client system Name as String.

7.1.6 Function Name: displayClientInfo() : String

Description: this function displays the Client Information.

Input: void

Output: Client Information as String.

7.1.7 Function Name: getContolOperation(ClientName : String, Control : Integer = 0)

Description: This function retrieves the Control Operation selected by the user..

Input: Client system Name and Control code.

Output: void

7.1.8 Function Name: getControlMsg() : String

Description: This function gets the message to be displayed at the Client from the user while executing the Control Operation

Input: void

Output: Message as a String

7.1.9 Function Name: doControl(ClientName : String, Operation : Integer = 0, Message : String) : Boolean

Description: This function executes the selected Control operation on the Client selected.

Input: Client Name, Operation to be performed and the message as strings.

Output: Draws the map.

7.1.10 Function Name: getMsgRecClient() : String

Description: This function gets the Clients to whom the message has to be sent.

Input: void

Output: Client names as String.

7.1.11 Function Name: getMessage() : String

Description: This function gets the message to be sent.

Input: void.

Output: Message as String.

7.1.12 Function Name: sendMsg(ClientName : String, Message : String) : Integer

Description: this function sends the message to the clients specified.

Input: Client name and the Message.

Output: Error Code that may result in delivering the message.

7.1.13 Function Name: storeClientSysInfo(ClientName : String, Info : String)

Description: this function stores the Client information received from the Clients.

Input: Client name and the Information as Strings.

Output: void.

7.2 Client Class:

7.2.1 Function Name: sendDesktop() : Image

Description: ***This function Sends the desktop Image to the RMS Class.***

Input: Void

Output: Desktop Image.

7.2.2 Function Name: collectSysInfo() : String

Description: This function collects the Client System information.

Input: void.

Output: Information collected as String.

7.2.3 Function Name: userLogOn() : Boolean

Description: This function gets whether a user is currently logged on at the Client system.

Input: void.

Output: True or False as boolean.

7.2.4 Function Name: sendSysInfo() : String

Description: This function sends the system information from Client.

Input: void.

Output: Information as String.

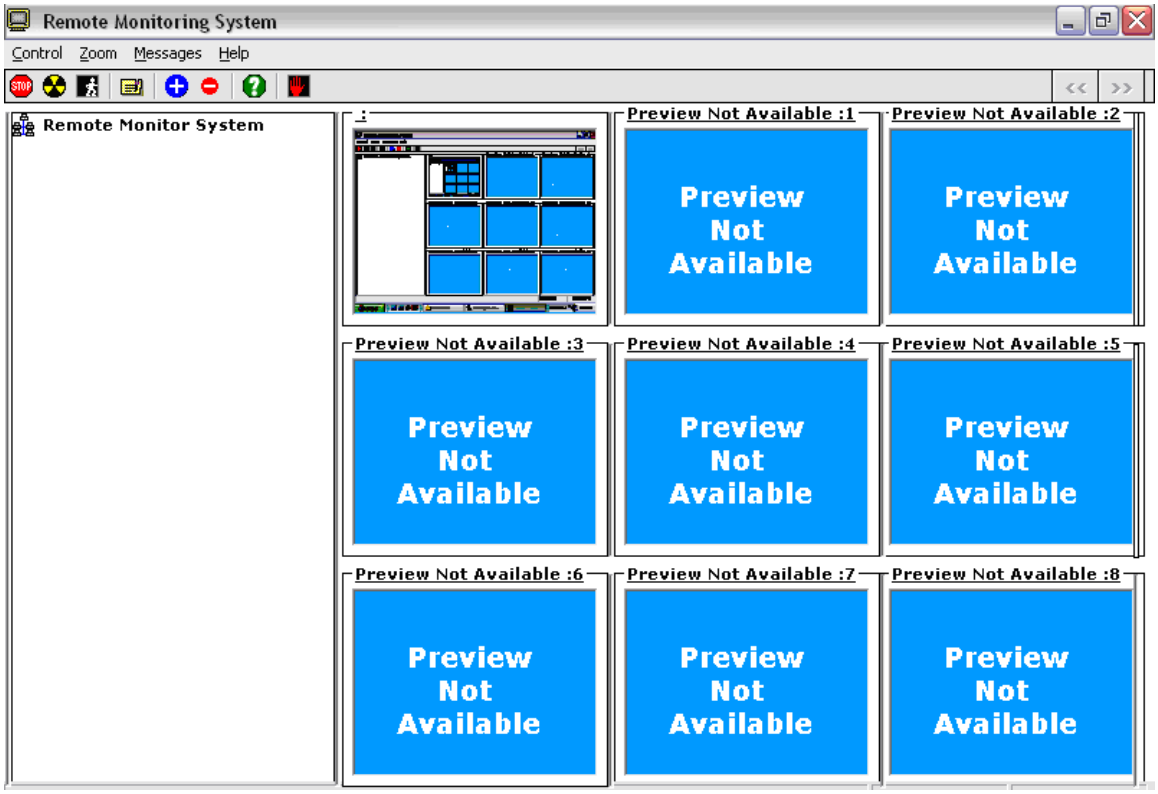
SCREENS

8. SCREENS

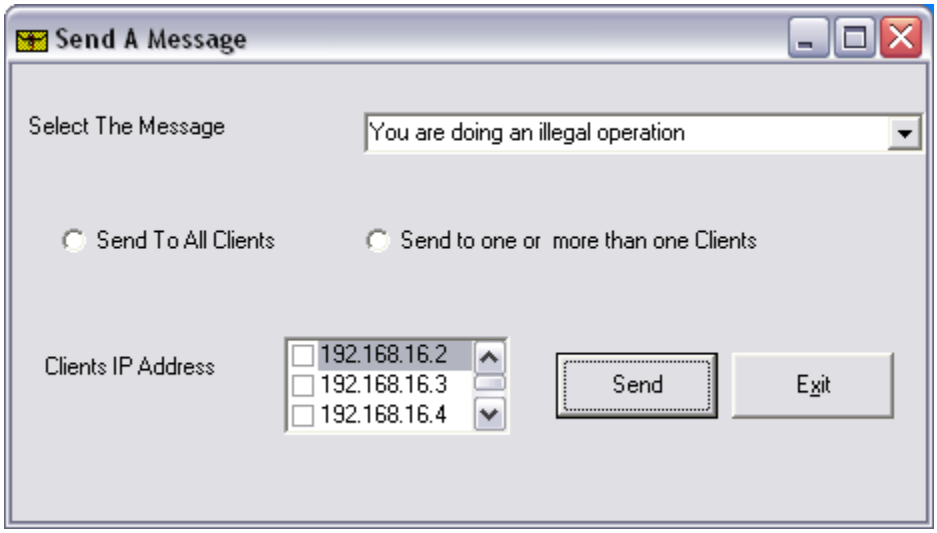
8.1 Introduction:

This section is intended to introduce the screens developed for the User Interface user can see at maximum of eight screens .

8.2 Main Screen:



8.3 Send Messages Screen:



8.4 Frame Full view Screen:

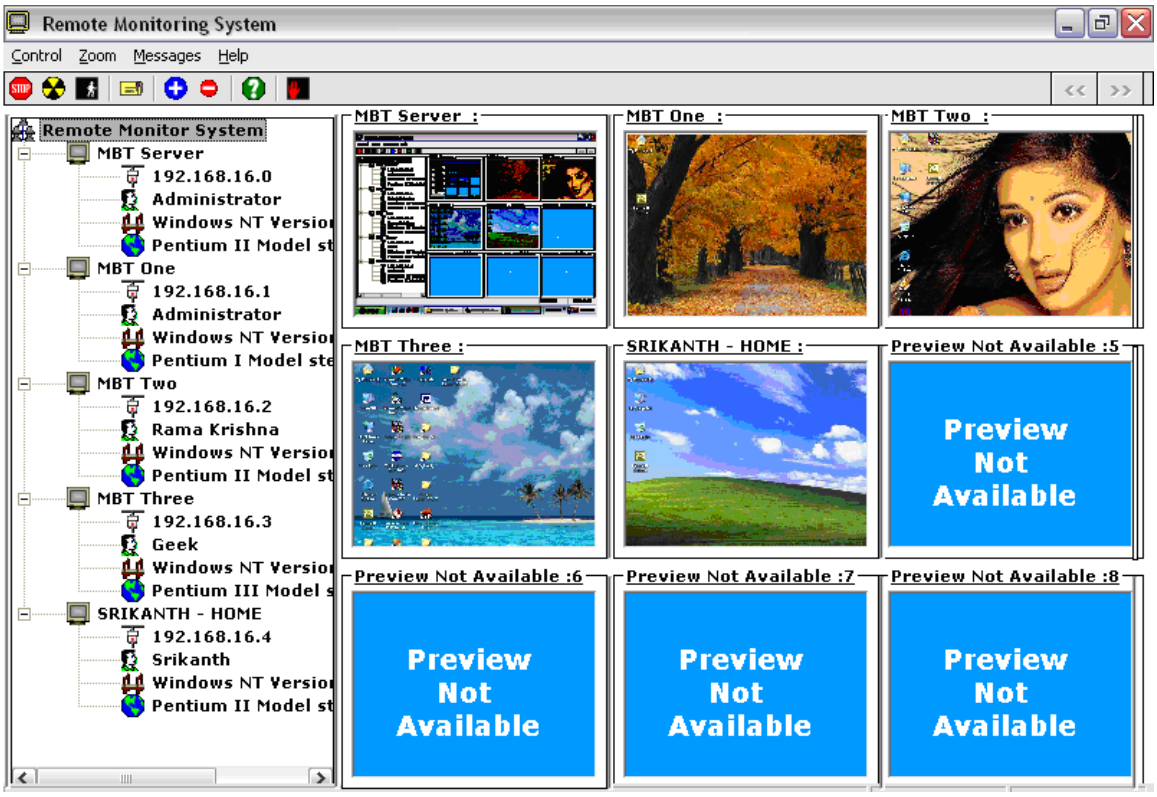
These are the Screens developed for the RMS front end using Microsoft Visual Studio 6.0.

TESTING

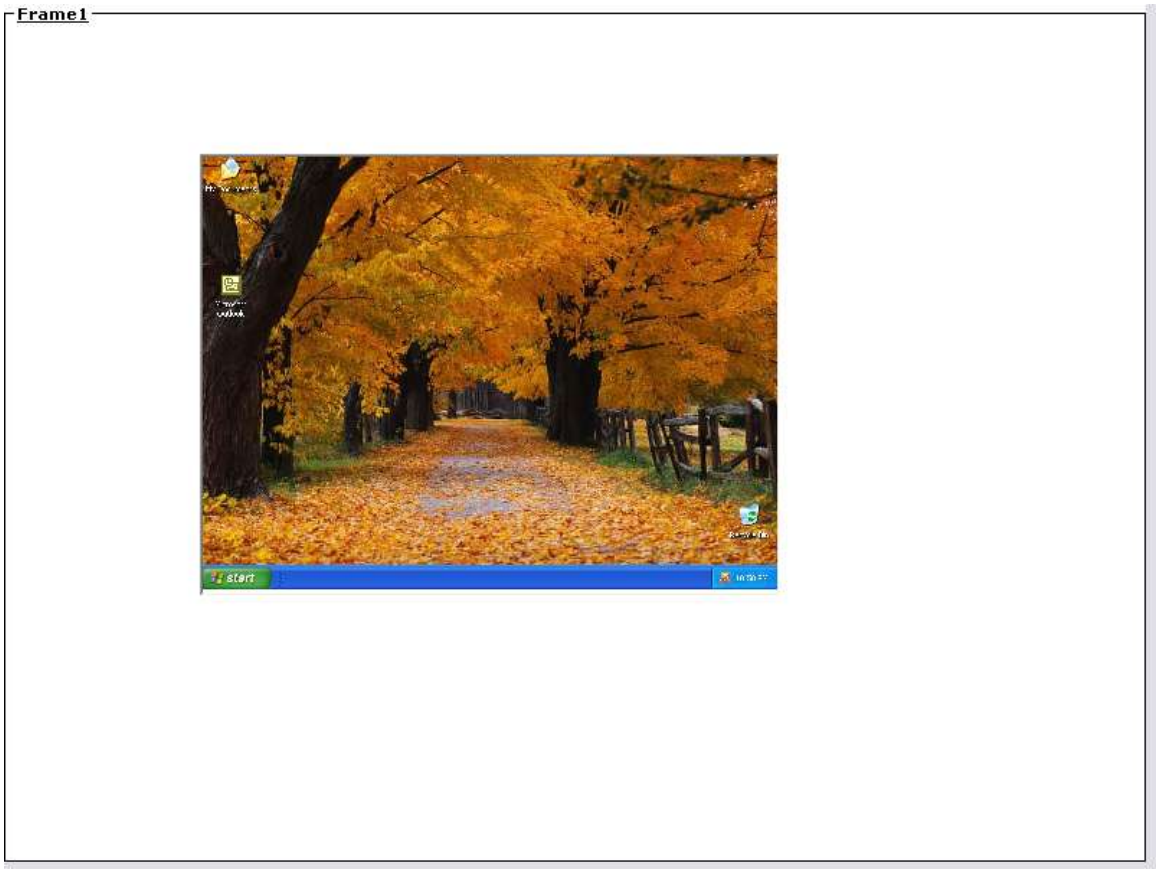
9. TESTING

9.1 Test Results:

Main Screen of the RMS with four Client systems(MBT one-MBT Two-MBT-three-srikanth home) connected to the Server (say MBT) and being Monitored using RMS is shown below(Facilities from International Systems Technologies Inc. were used her to get the following screens).



The Screen shot of a Client System after the Supervisor selected 50% zoom level:



All the functional features of RMS have been Tested Successfully.

CONCLUSION

&

FUTURE SCOPE

10. CONCLUSION AND FUTURE SCOPE OF WORK

10.1 Conclusion:

To develop a fully functional Remote Monitoring System we completed the required field study, analysis and design. The Remote Monitoring System is based on Rapid Application Development Paradigm and was designed using the Unified Modeling Language with the help of Rational Rose Tool. The components of the system were developed using Windows 32 Application Programming Interface in Visual Studio 6.0. All functional features of RMS have been successfully tested at Real time.

10.2 Future Scope of Work:

The use of Remote Monitoring System in Educational and Office Environments allows it to be launched as a Commercial Product. RMS can be made more powerful enhanced by enhancing its features for example, making it compliant to Cross platform environment, support for mobile Networks, and its performance can be improved by optimizing the utilization of Bandwidth more effectively.

BIBLIOGRAPHY

11. BIBLIOGRAPHY

- Mastering **UML** with RATIONAL ROSE
  -Wendy Boggs
  -Michael Boggs
- Integrated Approach to **Software Engineering**
  -Pankaj Jalote
- **Software Engineering** A Practitioner's approach
  -Roger S. Pressman
- www.rational.com/uml
- www.andreavb.com
- www.ambisoft.com
- www.experts-exchange.com
- www.remotecontrolssoftware.com
- www.bradapp.net
- www.campus.umr.edu/cis/desktop/
- www.microsoft.com/resources/documentation/Windows/XP/all/reskit/en-us/Default.asp
- www.remotelyanywhere.com